PVT - PHP Vulnerability Tracer

Documentation v 0.1., 25-10-2012
by Arthur Gerkis

Table of contents

1. Introduction

PVT is a PHP extension designed to make search of security flaws in PHP web-applications easier. As a tool for dynamic analysis, you can see what happens under the hood of web-application in a runtime.

2. Installation

Installation is done as follows:

```
$ tar -xzf pvt-0.1.x.tgz
$ cd pvt-0.1.x
$ phpize
$ ./configure
$ make
```

Then you have to add to php.ini PVT extension directives, provide your path to extension:

```
zend_extension=/var/www/PVT/modules/pvt.so
```

You can find pvt.ini example in folder with a source code. After PVT directives were added, restart your web-server. After that tool is ready to use - run web-application and watch logs.

PVT is able to run only on 32/64-bit Linux-based systems, 5.2/5.3 PHP versions. Work on other platforms is not guaranteed.

3. Logs structure

There are actually not so many files you would care about - just place folder logs/ where you want, make it writable and point to it:

```
pvt.log_file = /var/www/htdocs/PVT/logs
```

The important files and folders will be mentioned in current documentation.

PVT can write into different folders as well as to one:

```
pvt.log_one_folder = On
```

If logs are written to different folders, then there will be created folders showing script launch time, like pvt-2012-05-25_00:03:03/. In case if logs are written to folder with mode “w”, then the name of folder will be pvt-common-w/. When mode “a” is used, folder will be named as pvt-common-a/. Defined here:

```
pvt.log_write_mode = “a”
```

Note: 1) be sure to make folder writable. 2) in mode ‘a’ *.dot files will be appended too and that will break possibility to generate graphs.

4. Modules

You can switch on or off any modules you want - just define it in your php.ini file.

4.1. Opcode dumper

```
    pvt.dump_ops = On
```

Opcode dumper allows to see PHP web-application executed path in low-level - view opcodes were executed. All the logs are written to the folder opcodes/, where each filename is the name of appropriate calling function or a new file.
Note: this module will not work together with XCache extension.

4.2. Function tracer

```
    pvt.trace_func = On
```

Function tracer allows to trace all the functions that were executed during web-application runtime. It creates following files:

1. trace-functions.txt
2. trace-functions.dot

First one is a simply text file with raw dump of traces. Second file allows you to draw graphs. To do that, in dump/ folder you can find svg-mod.pl tool. Launch it as follows:

```
    $ perl svg-mod.pl <mode> <path/to/web-application/>
```

If everything goes well, you can open show.html and navigate generated svg file. More details about it you will find in a corresponding section “Using graph”.
Note: dot application has to be installed.

4.3. Evaluated strings hooker

Using this module you can observe which strings were evaluated. It includes following:

- eval(), assert(),
- preg_replace() with /e modifier
- create_function()

You can find results in dump-evaluated.phps file. It is possible to configure this module to catch some marker (string you want to find) and you can define how much characters around marker to log:

```
    pvt.eval_marker = ”your_marker”
    pvt.eval_hook_len = 256
```

If you supply zero, then it will log all strings unlimited.
To avoid duplicates, you can use following directive:

```
    pvt.eval_unique = On
```

You can override single marker hooking settings by switching on following directive:

```
    pvt.eval_hook_all = On
```

In this case will be hooked absolutely all evaluated strings (except of duplicates, if you want unique strings)

4.4. Variables dumper

```
	pvt.dump_vars = On
```

With this module variable contents will be dumped into dump-variables.txt. If you prefer to dump only some of variables, then you have to specify them:

```
	pvt.dump_vars_list = “var_1,...”
```

You can override previous setting to dump all variables:

```
	pvt.dump_vars_all = On
```

4.5. Substring search

```
	pvt.catch_marker = On
```

This module will write to the file catched-marker.txt all found substrings in function arguments. To specify which substring to look for, you have to set following directive:

```
	pvt.catch_marker_val = “marker”
```

Here only one marker is possible. To specify character length around substring to log, define it here:

```
	pvt.catch_marker_len = 256
```

If you supply zero, then it will log all strings unlimited.
It is also possible to list which functions to watch:

```
	pvt.catch_funcs = “foo,...”
```

Note: constructions like echo() will be missed.

5. Other features

5.1. Overriding settings

To avoid restarting server every time you want to change something in PVT settings, you can override them through GET or POST requests, by using special $_PVT variable. Example:

```
http://localhost/app/index.php?_PVT=dump_vars_all=0|dump_vars_list=page
```

Here were two following directives set:

```
	pvt.dump_vars_all= Off
	pvt.dump_vars_list = “page”
```

With pipe “|” you can list several different directives at one time.

5.2. Using graph

When you have launched show.html file, you should see graph. It may take a while to load SVG if analyzed web-application is heavy. Current graph options:

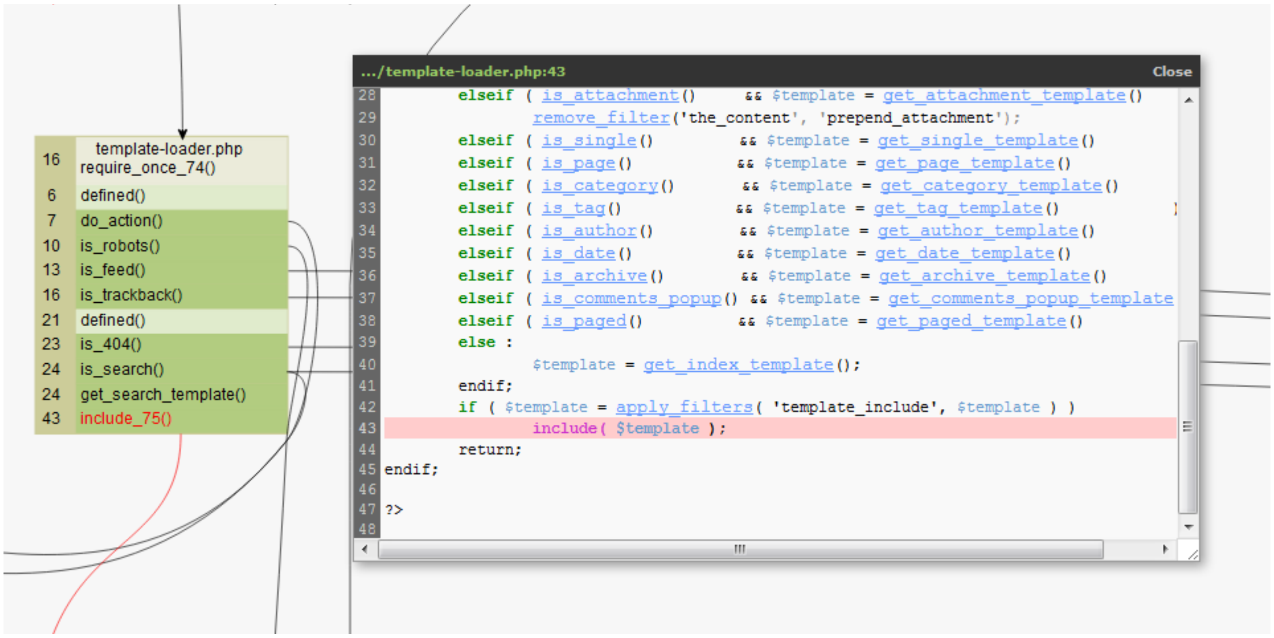
- click on function name - path to the next function block will be highlighted;
- click on function line number - file will be loaded with highlighted syntax;
- inside of loaded file you can click on functions - new window will open;
- dangerous functions will be automatically highlighted;
- you can highlight line inside opened window by clicking on it twice;

Note: include, require, eval, other constructions or functions that cause to load new php code, they will have numbers assigned like include_75(). This is temporary hack, in future it will be removed.

In web-applications quite often loops are encountered - this will cause to draw huge meshes in graph. To avoid that, you can use graph folding:

```
pvt.graph_folding = On
```

As an example of what you should see, here is depicted part of a Wordpress graph:



```
16  template-loader.php
    require_once_74()
6   defined()
7   do_action()
10  is_robots()
13  is_feed()
16  is_trackback()
21  defined()
23  is_404()
24  is_search()
24  get_search_template()
43  include_75()
```

```
        elseif ( is_attachment()    && $template = get_attachment_template() )
                remove_filter('the_content', 'prepend_attachment');
        elseif ( is_single()        && $template = get_single_template()
        elseif ( is_page()          && $template = get_page_template()
        elseif ( is_category()      && $template = get_category_template()
        elseif ( is_tag()           && $template = get_tag_template()
        elseif ( is_author()        && $template = get_author_template()
        elseif ( is_date()          && $template = get_date_template()
        elseif ( is_archive()       && $template = get_archive_template()
        elseif ( is_comments_popup() && $template = get_comments_popup_template
        elseif ( is_paged()         && $template = get_paged_template()
        else :
                $template = get_index_template();
        endif;
        if ( $template = apply_filters( 'template_include', $template ) )
                include( $template );
        return;
endif;

?>
```

5.3. Miscellaneous

PVT can count web-application statistics, it will be written to stats.txt. Following directive is responsible for that:

```
pvt.count_stats = On
```

To avoid using PVT in directories you don't want to, you can use .htaccess file that comes together with PVT.

6. Disclaimer

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Please note that this tool is not mentioned for use on production or any public servers.

7. Bugs

This project is still in early development stage, bugs are possible. Please report all the PVT bugs you have found.

If extension crashes, then you can submit script that triggers bug, or you can generate backtrace as follows:

1. install PHP symbols if you have not done that yet (package “php5-dbg”);
2. run your PHP script under debugger, like:
   a. `$ gdb php`
   b. `(gdb) run -f /var/www/script.php`
   c. `(gdb) bt`
3. submit generated backtrace (bt).

As well you can run valgrind, if you are suspicious about any memory problems:

```
$ valgrind --tool=memcheck --leak-check=full --num-callers=30 --track-origins=yes --log-file=./php.log php /var/www/script.php
```

Sure, you can also send backtrace/script directly to me, if you do not want to file a bug.